Model Predictive Control
A Step-by-Step Implementation Guide

Yajur Kumar

yajurkumar@outlook.com
www.linkedin.com/in/yajur

April 28, 2023

1 Introduction

Model Predictive Control (MPC) is an advanced control strategy that has been widely adopted in various industries due to its ability to handle multivariable systems with constraints, predict system behavior, and adjust control actions in real-time. This article presents a step-by-step implementation guide of Model Predictive Control, covering the essential concepts and implementation techniques.

2 Understand the Basics of Model Predictive Control

MPC uses a mathematical model of the system to predict its future behavior and calculate the optimal control actions accordingly. The fundamental concepts of MPC are:

1. System Model: A mathematical representation of the system's dynamics, typically represented as a set of differential or difference equations. For a linear system, the state-space model can be represented as:

$$x(k+1) = Ax(k) + Bu(k) \tag{1}$$

where $x(k)$ is the system state at time step k, $u(k)$ is the control input, A is the system matrix, and B is the input matrix.

2. Prediction Horizon: The time window over which the system's future behavior is predicted, denoted by N.

3. Control Horizon: The time window over which the control actions are optimized, denoted by M.

4. Optimization Problem: MPC solves an optimization problem at each control interval to determine the best control actions that minimize a cost function while satisfying system constraints.

3 Obtain or Develop a System Model

To implement MPC, you need to have a mathematical model that represents the system dynamics. You can obtain a model through first principles (physics-based modeling), system identification, or data-driven techniques, such as machine learning.

4 Define the Cost Function and Constraints

The cost function quantifies the control objectives and is minimized by the optimization algorithm. A typical cost function includes terms representing tracking error, control effort, and possibly constraints violations. Define the cost function based on your control objectives and system requirements. A common quadratic cost function is:

$$J = \sum_{k=0}^{N-1} [(x(k) - x_{ref}(k))^T Q(x(k) - x_{ref}(k)) + u(k)^T R u(k)] \tag{2}$$

where Q and R are positive definite weighting matrices, and $x_{ref}(k)$ is the reference state at time step k.

Additionally, identify the constraints on the system states and control actions. These constraints can be in the form of bounds, rate limits, or other relationships between the variables.

5 Choose an Optimization Algorithm

Select an appropriate optimization algorithm to solve the optimization problem. Commonly used optimization algorithms for MPC include:

1. Quadratic Programming (QP)

2. Sequential Quadratic Programming (SQP)

3. Interior Point Methods

The choice of the algorithm depends on the nature of the problem, such as whether it is linear or nonlinear, and the available computational resources.

6 Implement the MPC Algorithm

Implement the MPC algorithm in a programming language or environment of your choice. The key steps in the MPC algorithm are:

1. At each control interval, obtain the current system state.

2. Solve the optimization problem using the current system state, system model, cost function, constraints, and optimization algorithm.

3. Apply the first control action from the optimal control sequence to the system.

4. Move to the next control interval and repeat the process.

7 Validate and Fine-Tune the MPC Controller

Validate the performance of the MPC controller by simulating or testing it on the actual system. Compare the controller's performance to the desired control objectives and adjust the cost function, prediction horizon, or control horizon if necessary. Fine-tuning the controller may involve iterative trials and adjustments to achieve the desired performance.

8 Integrate the MPC Controller into the System

Once the MPC controller's performance is satisfactory, integrate it into the system. This may involve embedding the MPC algorithm into a microcontroller, PLC, or real-time computer, depending on the system requirements.

9 Example: Inverted Pendulum in a Cart

The state space model of an inverted pendulum in a moving cart can be described as:

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \\ \dot{x}_4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & -\frac{mg}{M} & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & \frac{(M+m)g}{Ml} & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{M} \\ 0 \\ -\frac{1}{Ml} \end{bmatrix} u \tag{3}$$

where, x_1, x_2, x_3, and x_4 represent the cart position, cart velocity, pendulum angle, and pendulum angular velocity, respectively. The parameters m, M, l, and g denote the mass of the pendulum, mass of the cart, length of the pendulum, and acceleration due to gravity, respectively. The variable u represents the control input applied to the cart.

The provided MATLAB code demonstrates controlling an inverted pendulum using the MPC control strategy. The code simulates the pendulum dynamics, designs an MPC controller, and animates the pendulum and cart system.

```
1
2  % (c) Yajur Kumar, 2023. All rights reserved.
3  % Program to simulate control of inverted pendulum
4  % using Model Predictive Control
5
6  clear; close all; clc;
7  %% Inverted Pendulum Model Parameters
8  M = 5.0; % Mass of the cart [kg]
9  m = 2.0; % Mass of the pendulum [kg]
10 b = 0.1; % Coefficient of friction for cart [N/m/s]
11 l = 0.5; % Length to pendulum center of mass [m]
12 g = 9.81; % Gravity [m/s^2]
13
14 %% Linearized state-space model
15 A = [0 1 0 0;
16     0 -b/M -m*g/M 0;
17     0 0 0 1;
18     0 -b/(M*l) -(m+M)*g/(M*l) 0];
19
20 B = [0; 1/M; 0; 1/(M*l)];
21
22 C = [1 0 0 0;
23     0 0 1 0];
24
25 D = [0; 0];
26
27 %% Discretize the system
```

```
Ts = 0.1; % Sampling time [s]
sys = ss(A, B, C, D);
sys_d = c2d(sys, Ts, 'zoh'); % Discretized system
Ad = sys_d.a;
Bd = sys_d.b;
Cd = sys_d.c;

%% Design MPC Controller
N = 20; % Prediction horizon
Nu = 5; % Control horizon
Q = diag([10, 1, 100, 1]); % State weighting matrix
R = 0.1; % Control input weighting

% Create MPC controller object
mpcobj = mpc(sys_d, Ts);
mpcobj.PredictionHorizon = N;
mpcobj.ControlHorizon = Nu;
mpcobj.Weights.OutputVariables = [10, 100];
mpcobj.Weights.ManipulatedVariables = R;
mpcobj.Weights.ManipulatedVariablesRate = 0.1;

% Constraints
mpcobj.MV(1).Min = -10; % Force min [N]
mpcobj.MV(1).Max = 10; % Force max [N]

%% Simulation
Nsim = 100; % Number of simulation steps
x0 = [0.2; 0; 0.1; 0]; % Initial state

% Preallocate arrays for simulation results
nx = size(Ad, 1);
nu = size(Bd, 2);
ny = size(Cd, 1);
x = zeros(nx, Nsim);
y = zeros(ny, Nsim);
u = zeros(nu, Nsim);
x(:,1) = x0;
y(:,1) = Cd * x0;

% Create initial MPC state
mpcstate = mpcstate(mpcobj, x(:,1));

% Main simulation loop
for k = 1:Nsim-1
    % Measure the current output
    ymeas = y(:,k);

    % Compute the control action using the MPC controller
    [u(:,k), mpc_info] = mpcmove(mpcobj, mpcstate, ymeas, []);
```

```
77
78      % Apply the control action to the system
79      % and obtain the next state
80      x(:,k+1) = Ad * x(:,k) + Bd * u(:,k);
81
82      % Calculate the output
83      y(:,k+1) = Cd * x(:,k+1);
84
85      % Update MPC state
86      mpcstate.Plant = x(:,k+1);
87  end
88
89  % Plot the results
90  t = (0:Nsim-1) * Ts;
91  figure(1);
92  subplot(3,1,1);
93  plot(t, y(1,:), 'LineWidth', 2);
94  ylabel('Cart␣Position␣(m)');
95  title('Inverted␣Pendulum␣MPC␣Control');
96  grid on;
97
98  subplot(3,1,2);
99  plot(t, y(2,:), 'LineWidth', 2);
100 ylabel('Pendulum␣Angle␣(rad)');
101 grid on;
102
103 subplot(3,1,3);
104 plot(t, u(:,1:Nsim), 'LineWidth', 2);
105 xlabel('Time␣(s)');
106 ylabel('Force␣(N)');
107 grid on;
108
109 %% Animate Pendulum
110 figure(2);
111 xlim([-1, 1]);
112 ylim([-1, 1]);
113 hold on;
114
115 % Define cart and pendulum parameters
116 cart_width = 0.2;
117 cart_height = 0.1;
118
119 for k = 1:Nsim
120     cla; % Clear current axes
121     cart_x = y(1,k) - cart_width / 2;
122     cart_y = -cart_height / 2;
123
124     pendulum_x = [y(1,k), y(1,k) + l * sin(y(2,k))];
125     pendulum_y = [0, -l * cos(y(2,k))];
```

```
126
127     % Draw cart
128     rectangle('Position', [cart_x, cart_y, cart_width,
129 cart_height], 'FaceColor', 'b');
130
131     % Draw pendulum
132     plot(pendulum_x, -pendulum_y, 'r', 'LineWidth', 2);
133     plot(pendulum_x(2), -pendulum_y(2), 'ro', 'MarkerSize',
134 10, 'MarkerFaceColor', 'r');
135
136     % Draw rail
137     line([-1, 1], [-cart_height/2, -cart_height/2], 'Color',
138 'k', 'LineWidth', 1);
139
140     % Set axis limits and labels
141     xlim([-1, 1]);
142     ylim([-1, 1]);
143     xlabel('Position␣(m)');
144     ylabel('Height␣(m)');
145     title('Inverted␣Pendulum␣Animation');
146
147     drawnow;
148     pause(0.05);
149 end
```

10 Conclusion

Model Predictive Control is a powerful control strategy that has proven its effectiveness in various applications, such as controlling an inverted pendulum. Implementing MPC involves understanding the basic concepts, developing a system model, defining the cost function and constraints, selecting an optimization algorithm, implementing the MPC algorithm, and fine-tuning the controller. By following this step-by-step guide and the provided MATLAB code example, you can successfully implement an MPC controller for your system and enjoy its numerous benefits.

References

1. Maciejowski, J. M. (2002). Predictive Control with Constraints. Prentice Hall.

2. Rawlings, J. B., & Mayne, D. Q. (2009). Model Predictive Control: Theory and Design. Nob Hill Publishing.

3. Borrelli, F., Bemporad, A., & Morari, M. (2017). Predictive Control for Linear and Hybrid Systems. Cambridge University Press.

4. MATLAB (2021). Model Predictive Control Toolbox User's Guide. The MathWorks, Inc.

5. Åström, K. J., & Murray, R. M. (2010). Feedback Systems: An Introduction for Scientists and Engineers. Princeton University Press.